Microsoft Knowledge Base Article - 247412

INFO: Methods for Transferring Data to Excel from Visual Basic

Applies To

This article was previously published under Q247412

SUMMARY

This article discusses numerous methods for transferring data to Microsoft Excel from your Microsoft Visual Basic application. This article also presents the advantages and the disadvantages for each method so that you can choose the solution that works best for you.

MORE INFORMATION

The approach most commonly used to transfer data to an Excel workbook is Automation. Automation gives you the greatest flexibility for specifying the location of your data in the workbook as well as the ability to format the workbook and make various settings at run time. With Automation, you can use several approaches for transferring your data:

- Transfer data cell by cell
- Transfer data in an array to a range of cells
- Transfer data in an ADO recordset to a range of cells using the **CopyFromRecordset** method
- Create a **QueryTable** on an Excel worksheet that contains the result of a query on an ODBC or OLEDB data source
- Transfer data to the clipboard and then paste the clipboard contents into an Excel worksheet

There are also methods that you can use to transfer data to Excel that do not necessarily require Automation. If you are running an application server-side, this can be a good approach for taking the bulk of processing the data away from your clients. The following methods can be used to transfer your data without Automation:

- Transfer your data to a tab- or comma-delimited text file that Excel can later parse into cells on a worksheet
- Transfer your data to a worksheet using ADO
- Transfer data to Excel using Dynamic Data Exchange (DDE)

The following sections provide more detail on each of these solutions.

Use Automation to Transfer Data Cell by Cell

With Automation, you can transfer data to a worksheet one cell at a time:

```
Dim oExcel As Object
Dim oBook As Object
Dim oSheet As Object

'Start a new workbook in Excel
Set oExcel = CreateObject("Excel.Application")
Set oBook = oExcel.Workbooks.Add


'Add data to cells of the first worksheet in the new workbook
Set oSheet = oBook.Worksheets(1)
oSheet.Range("A1").Value = "Last Name"
oSheet.Range("B1").Value = "First Name"
oSheet.Range("A1:B1").Font.Bold = True
oSheet.Range("A2").Value = "Doe"
oSheet.Range("B2").Value = "John"

'Save the Workbook and Quit Excel
oBook.SaveAs "C:\Book1.xls"
oExcel.Quit
```

Transferring data cell by cell can be a perfectly acceptable approach if the amount of data is small. You have the flexibility to place data anywhere in the workbook and can format the cells conditionally at run time. However, this approach is **not** recommended if you have a large amount of data to transfer to an Excel workbook. Each **Range** object that you acquire at run time results in an interface request so that transferring data in this manner can be slow. Additionally, Microsoft Windows 95 and Windows 98 have a 64K limitation on interface requests. If you reach or exceed this 64k limit on interface requests, the Automation server (Excel) might stop responding or you might receive errors indicating low memory. This limitation for Windows 95 and Windows 98 is discussed in the following Knowledge Base article:

216400 PRB: Cross-Process COM Automation Can Hang Client Application on Win95/98

Once more, transferring data cell by cell is acceptable only for small amounts of data. If you need to transfer large data sets to Excel, you should consider one of the solutions presented later .

For more sample code for Automating Excel, please see the following article in the Microsoft Knowledge Base:

219151 HOWTO: Automate Microsoft Excel from Visual Basic

Use Automation to Transfer an Array of Data to a Range on a Worksheet

An array of data can be transferred to a range of multiple cells at once:

```
Dim oExcel As Object
Dim oBook As Object
Dim oSheet As Object

'Start a new workbook in Excel
Set oExcel = CreateObject("Excel.Application")
Set oBook = oExcel.Workbooks.Add

'Create an array with 3 columns and 100 rows
Dim DataArray(1 To 100, 1 To 3) As Variant
Dim r As Integer
For r = 1 To 100
   DataArray(r, 1) = "ORD" & Format(r, "0000")
   DataArray(r, 2) = Rnd() * 1000
   DataArray(r, 3) = DataArray(r, 2) * 0.7
Next

'Add headers to the worksheet on row 1
Set oSheet = oBook.Worksheets(1)
oSheet.Range("A1:C1").Value = Array("Order ID", "Amount", "Tax")

'Transfer the array to the worksheet starting at cell A2
oSheet.Range("A2").Resize(100, 3).Value = DataArray

'Save the Workbook and Quit Excel
oBook.SaveAs "C:\Book1.xls"
oExcel.Quit
```

If you transfer your data using an array rather than cell by cell, you can realize an enormous performance gain with a large amount of data. Consider this line from the code above that transfers data to 300 cells in the worksheet:

```
oSheet.Range("A2").Resize(100, 3).Value = DataArray
```

This line represents two interface requests (one for the **Range** object that the **Range** method returns and another for the **Range** object that the **Resize** method returns). On the other hand, transferring the data cell by cell would require requests for 300 interfaces to **Range** objects. Whenever possible, you can benefit from transferring your data in bulk and reducing the number of interface requests you make.

Use Automation to Transfer an ADO Recordset to a Worksheet Range

Excel 2000 introduced the **CopyFromRecordset** method that allows you to transfer an ADO (or DAO) recordset to a range on a worksheet. The following code illustrates how you could automate Excel 2000, Excel 2002, or Office Excel 2003 and transfer the contents of the Orders table in the Northwind Sample Database using the **CopyFromRecordset** method:

```
'Create a Recordset from all the records in the Orders table
Dim sNWind As String
Dim conn As New ADODB.Connection
Dim rs As ADODB.Recordset
sNWind = _
   "C:\Program Files\Microsoft Office\Office\Samples\Northwind.mdb"
conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
   sNWind & ";"
conn.CursorLocation = adUseClient
Set rs = conn.Execute("Orders", , adCmdTable)

'Create a new workbook in Excel
Dim oExcel As Object
Dim oBook As Object
Dim oSheet As Object
Set oExcel = CreateObject("Excel.Application")
Set oBook = oExcel.Workbooks.Add
Set oSheet = oBook.Worksheets(1)

'Transfer the data to Excel
oSheet.Range("A1").CopyFromRecordset rs

'Save the Workbook and Quit Excel
oBook.SaveAs "C:\Book1.xls"
oExcel.Quit

'Close the connection
rs.Close
conn.Close
```

Excel 97 also provides a **CopyFromRecordset** method but you can use it only with a DAO recordset. **CopyFromRecordset** with Excel 97 does not support ADO.

For more information about using ADO and the **CopyFromRecordset** method, please see the following article in the Microsoft Knowledge Base:

246335 HOWTO: Transfer Data from ADO Recordset to Excel with Automation

Use Automation to Create a QueryTable on a Worksheet

A **QueryTable** object represents a table built from data returned from an external data source. While automating Microsoft Excel, you can create a **QueryTable** by simply providing a connection string to an OLEDB or an ODBC data source along with an SQL string. Excel assumes the responsibility for generating the recordset and inserting it into the worksheet at the location you specify. Using **QueryTables** offers several advantages over the **CopyFromRecordset** method:

- Excel handles the creation of the recordset and its placement into the worksheet.
- The query can be saved with the **QueryTable** so that it can be refreshed at a later time to obtain an updated recordset.
- When a new **QueryTable** is added to your worksheet, you can specify that data already existing in cells on the worksheet be shifted to accommodate the new data (see the **RefreshStyle** property for details).

The following code demonstrates how you could automate Excel 2000, Excel 2002, or Office Excel 2003 to create a new **QueryTable** in an Excel worksheet using data from the Northwind Sample Database:

```
'Create a new workbook in Excel
Dim oExcel As Object
Dim oBook As Object
Dim oSheet As Object
Set oExcel = CreateObject("Excel.Application")
Set oBook = oExcel.Workbooks.Add
Set oSheet = oBook.Worksheets(1)

'Create the QueryTable
Dim sNWind As String
sNWind = _
   "C:\Program Files\Microsoft Office\Office\Samples\Northwind.mdb"
Dim oQryTable As Object
Set oQryTable = oSheet.QueryTables.Add( _
"OLEDB;Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
   sNWind & ";", oSheet.Range("A1"), "Select * from Orders")
oQryTable.RefreshStyle = xlInsertEntireRows
oQryTable.Refresh False

'Save the Workbook and Quit Excel
oBook.SaveAs "C:\Book1.xls"
oExcel.Quit
```

Use the Clipboard

The Windows Clipboard can also be used as a mechanism for transferring data to a worksheet. To paste data into multiple cells on a worksheet, you can copy a string where columns are delimited by tab characters and rows are delimited by carriage returns. The following code illustrates how Visual Basic can use its Clipboard object to transfer data to Excel:

```
'Copy a string to the clipboard
Dim sData As String
sData = "FirstName" & vbTab & "LastName" & vbTab & "Birthdate" & vbCr _
        & "Bill" & vbTab & "Brown" & vbTab & "2/5/85" & vbCr _
        & "Joe" & vbTab & "Thomas" & vbTab & "1/1/91"
Clipboard.Clear

Clipboard.SetText sData

'Create a new workbook in Excel
Dim oExcel As Object
Dim oBook As Object
Set oExcel = CreateObject("Excel.Application")
Set oBook = oExcel.Workbooks.Add


'Paste the data
oBook.Worksheets(1).Range("A1").Select
oBook.Worksheets(1).Paste

'Save the Workbook and Quit Excel
oBook.SaveAs "C:\Book1.xls"
oExcel.Quit
```

Create a Delimited Text File that Excel Can Parse into Rows and Columns

Excel can open tab- or comma-delimited files and correctly parse the data into cells. You can take advantage of this feature when you want to transfer a large amount of data to a worksheet while using little, if any, Automation. This might be a good approach for a client-server application because the text file can be generated server-side. You can then open the text file at the client, using Automation where it is appropriate.

The following code illustrates how you can create a comma-delimited text file from an ADO recordset:

```
'Create a Recordset from all the records in the Orders table
Dim sNWind As String
Dim conn As New ADODB.Connection
Dim rs As ADODB.Recordset
Dim sData As String
sNWind = _
   "C:\Program Files\Microsoft Office\Office\Samples\Northwind.mdb"
conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
   sNWind & ";"
conn.CursorLocation = adUseClient
Set rs = conn.Execute("Orders", , adCmdTable)

'Save the recordset as a tab-delimited file
sData = rs.GetString(adClipString, , vbTab, vbCr, vbNullString)
Open "C:\Test.txt" For Output As #1
Print #1, sData
Close #1

'Close the connection
rs.Close
conn.Close

'Open the new text file in Excel
Shell "C:\Program Files\Microsoft Office\Office\Excel.exe " & _
   Chr(34) & "C:\Test.txt" & Chr(34), vbMaximizedFocus
```

If your text file has a .CSV extension, Excel opens the file without displaying the Text Import Wizard and automatically assumes that the file is comma-delimited. Similarly, if your file has a .TXT extension, Excel automatically parse the file using tab delimiters.

In the previous code sample, Excel was launched using the **Shell** statement and the name of the file was used as a command line argument. No Automation was used in the previous sample. However, if so desired, you could use a minimal amount of Automation to open the text file and save it in the Excel workbook format:

```
'Create a new instance of Excel
Dim oExcel As Object
Dim oBook As Object
Dim oSheet As Object
Set oExcel = CreateObject("Excel.Application")

'Open the text file
Set oBook = oExcel.Workbooks.Open("C:\Test.txt")

'Save as Excel workbook and Quit Excel
oBook.SaveAs "C:\Book1.xls", xlWorkbookNormal
oExcel.Quit
```

For more information about using File I/O from your Visual Basic application, please see the following article in the Microsoft Knowledge Base:

172267 RECEDIT.VBP Demonstrates File I/O in Visual Basic

The following article also provides a discussion and sample code for controlling File I/O with Visual Basic for Applications:

File Access with Visual Basic for Applications

Transfer Data to a Worksheet Using ADO

Using the Microsoft Jet OLE DB Provider, you can add records to a table in an existing Excel workbook. A "table" in Excel is merely a range with a defined name. The first row of the range must contain the headers (or field names) and all subsequent rows contain the records. The following steps illustrate how you can create a workbook with an empty table named **MyTable**:

1. Start a new workbook in Excel.
2. Add the following headers to cells A1:B1 of Sheet1:

   A1: FirstName B1: LastName

3. Format cell B1 as right-aligned.
4. Select A1:B1.
5. On the **Insert** menu, choose **Names** and then select **Define**. Enter the name **MyTable** and click **OK**.
6. Save the new workbook as C:\Book1.xls and quit Excel.

To add records to **MyTable** using ADO, you can use code similar to the following:

```
'Create a new connection object for Book1.xls
Dim conn As New ADODB.Connection
conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
   "Data Source=C:\Book1.xls;Extended Properties=Excel 8.0;"
conn.Execute "Insert into MyTable (FirstName, LastName)" & _
   " values ('Bill', 'Brown')"
conn.Execute "Insert into MyTable (FirstName, LastName)" & _
   " values ('Joe', 'Thomas')"
conn.Close
```

When you add records to the table in this manner, the formatting in the workbook is maintained. In the previous example, new fields added to column B are formatted with right alignment. Each record that is added to a row borrows the format from the row above it.

You should note that when a record is added to a cell or cells in the worksheet, it overwrites any data previously in those cells; in other words, rows in the worksheet are not "pushed down" when new records are added. You should keep this in mind when designing the layout of data on your worksheets.

For additional information on using ADO to access an Excel workbook, please see the following articles in the Microsoft Knowledge Base:

195951 HOWTO: Query and Update Excel Data Using ADO From ASP

Use DDE to Transfer Data to Excel

DDE is an alternative to Automation as a means for communicating with Excel and transferring data; however, with the advent of Automation and COM, DDE is no longer the preferred method for communicating with other applications and should only be used when there is no other solution available to you.

To transfer data to Excel using DDE, you can:

- Use the **LinkPoke** method to poke data to a specific range of cell(s),

  -or-
- Use the **LinkExecute** method to send commands that Excel will execute.

The following code example illustrates how to establish a DDE conversation with Excel so that you can poke data to cells on a worksheet and execute commands. Using this sample, for a DDE conversation to be successfully established to the **LinkTopic** Excel|MyBook.xls, a workbook with the name **MyBook.xls** must already be opened in a running instance of Excel.

NOTE: In this example, **Text1** represents a **Text Box** control on a Visual Basic form:

```
'Initiate a DDE communication with Excel
Text1.LinkMode = 0
Text1.LinkTopic = "Excel|MyBook.xls"
Text1.LinkItem = "R1C1:R2C3"
Text1.LinkMode = 1

'Poke the text in Text1 to the R1C1:R2C3 in MyBook.xls
Text1.Text = "one" & vbTab & "two" & vbTab & "three" & vbCr & _
             "four" & vbTab & "five" & vbTab & "six"
Text1.LinkPoke

'Execute commands to select cell A1 (same as R1C1) and change the font
'format
```

```
        Text1.LinkExecute "[SELECT(""R1C1"")]"
        Text1.LinkExecute "[FONT.PROPERTIES(""Times New Roman"",""Bold"",10)]"

        'Terminate the DDE communication
        Text1.LinkMode = 0
```

When using **LinkPoke** with Excel, you specify the range in row-column (R1C1) notation for the **LinkItem**. If you are poking data to multiple cells, you can use a string where the columns are delimited by tabs and rows are delimited by carriage returns.

When you use **LinkExecute** to ask Excel to carry out a command, you must give Excel the command in the syntax of the Excel Macro Language (XLM). The XLM documentation is not included with Excel versions 97 and later. For more information on how you can obtain the XLM documentation, please see the following article in the Microsoft Knowledge Base:

143466 Download the Excel 97 Macro Functions Help File for XLM Documentation

DDE is not a recommended solution for communicating with Excel. Automation provides the greatest flexibility and gives you more access to the new features that Excel has to offer.

REFERENCES

For additional information, click the following article number to view the article in the Microsoft Knowledge Base:

306022 HOW TO: Transfer Data to an Excel Workbook by Using Visual Basic .NET

The information in this article applies to:

- Microsoft Office Excel 2003
- Microsoft Excel 2002
- Microsoft Excel 2000
- Microsoft Excel 97 for Windows
- Microsoft Visual Basic for Applications 5.0
- Microsoft Visual Basic for Applications 6.0

Last Reviewed: 12/12/2003 (5.0)

Keywords: kbAutomation kbDDE kbinfo KB247412 kbAudDeveloper